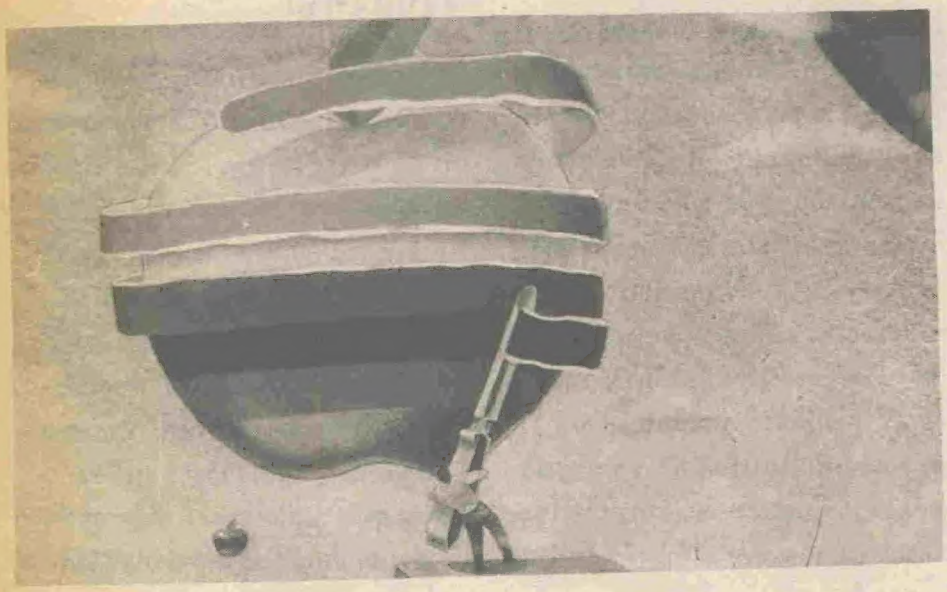
เรื่องน่าอ่าน
สมชาย นำประเสริฐชัย

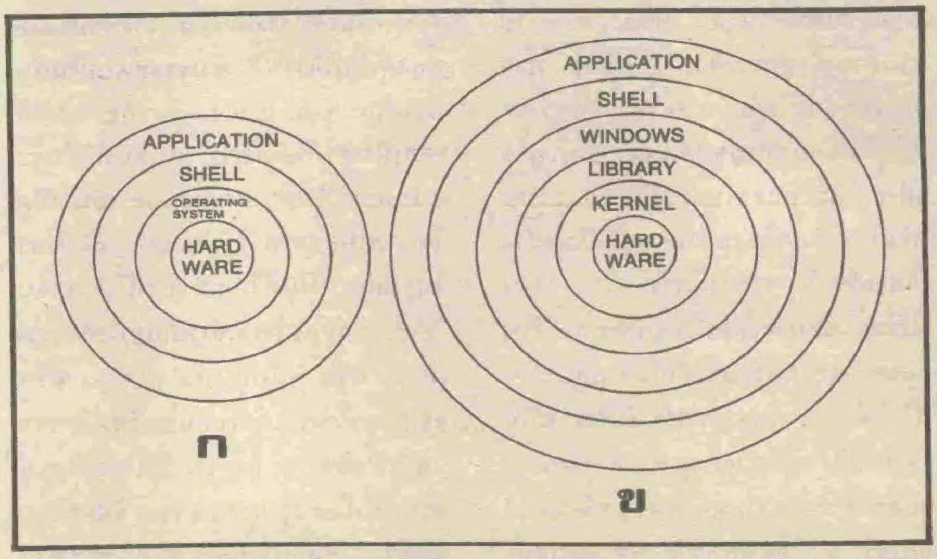
สารพัน...UNIX
ตอน เชลล์...ล่ามผู้ปราดเปรื่อง



ศึกษาความเป็นมาของเชลล์ การทำงาน รูปแบบคำสั่ง และเชลล์โปรแกรมมิง

เมื่อตอนที่แล้วได้กล่าวถึงความเป็นมาของระบบยูนิกซ์ เห็นได้ว่าสิ่งหนึ่งที่ทำให้ระบบยูนิกซ์มีความคล่องตัว ก็คือ ลักษณะโครงสร้างที่ได้รับการออกแบบมาเป็นระดับชั้นวงแหวน โดยมีเส้นกลางหรือแกนเป็นส่วนของฮาร์ดแวร์ ในส่วนนี้มีโปรแกรมจัดการเรียกว่า เคอร์เนล (Kernel) ซึ่งจะมีซอฟต์แวร์ระบบ (System Software) และบรรดาโปรแกรมต่าง ๆ อยู่รอบ ๆ ทำให้ผู้ใช้ไม่สามารถติดต่อกับส่วนของเคอร์เนลได้โดยตรง จำเป็นต้องผ่านตัวกลางคือ เชลล์

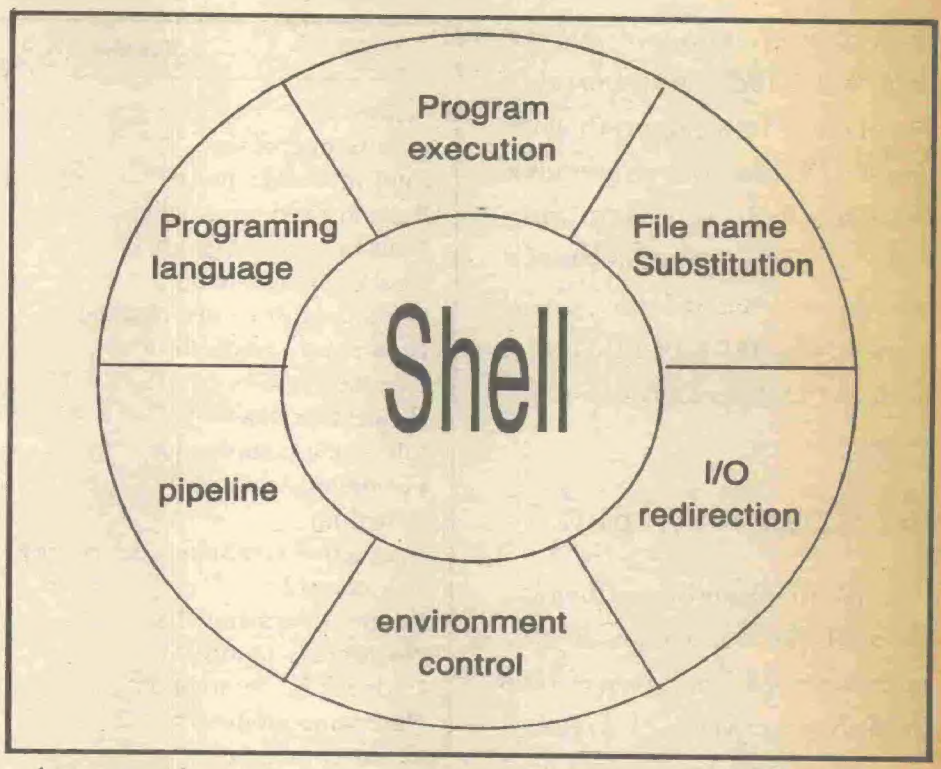
เชลล์เป็นส่วนที่เอื้ออำนวยให้การใช้ยูทิลิตี้ต่าง ๆ ได้สะดวกยิ่งขึ้น ชื่อของเชลล์ถือกำเนิดมาจากลักษณะโครงสร้างของระบบยูนิกซ์เอง และอยู่ในชั้นวงนอก ดังนั้นเชลล์จึงถือเป็นสภาวะหรือโปรแกรมเริ่มต้นในการใช้งาน และในตัวของระบบยูนิกซ์เองนั้นไม่ถือว่าเชลล์



รูปที่ 1 (ก) แสดงถึงโครงสร้างของระบบยูนิกซ์แบบเดิม โดยมีลักษณะเป็นชั้นวงแหวนที่มีการออกแบบมาจากพื้นฐานของระบบ MULTRICS ซึ่งเป็นจุดเริ่มต้นของระบบยูนิกซ์นั่นเอง
(ข) แสดงลักษณะโครงสร้างของระบบยูนิกซ์ที่ได้รับการพัฒนา โดยส่วนของ เคอร์เนลได้รับการเปลี่ยนแปลงเพื่อให้รองรับการดำเนินงานแบบมัลติโปรเซสซิง อีกทั้งมีส่วนของไลบรารีและวินโดว์เพิ่มขึ้น

เป็นส่วนหนึ่งของระบบปฏิบัติการ การเปลี่ยนแปลงของเชลล์นั้นมีการเปลี่ยนแปลงพัฒนาควบคู่ไปกับระบบยูนิกซ์มาตลอด จนในปัจจุบันนี้มีเชลล์หลายชนิด

เชลล์เป็นตัวเชื่อมระหว่างผู้ใช้กับทรัพยากรต่าง ๆ ของระบบ เช่น ไฟล์ และอุปกรณ์ต่าง ๆ โดยที่เชลล์จะทำหน้าที่ในรูปแบบของตัวแปรคำสั่ง (Command Interpreter) โดยผู้ใช้สามารถพิมพ์คำสั่ง แล้วเชลล์จะทำการตีความหมายและส่งคำสั่งเหล่านั้นออกไป และในเชลล์หลายชนิดยังได้เพิ่มเอดิเตอร์สำหรับเขียนคำสั่งเป็นขบวนความเข้ามาอีก เพื่อให้ผู้ใช้สามารถแก้ไขและเขียนคำสั่งใหม่ได้สะดวกยิ่งขึ้น ยังมีเชลล์หลายชนิดที่สามารถใช้ในรูปแบบของภาษาโปรแกรมได้โดยการนำเอาคำสั่งต่าง ๆ มาเรียงต่อเข้าด้วยกัน โดยลักษณะของโปรแกรมที่เขียนนี้จะรู้จักกันในนามของเชลล์สคริปต์ (Shell Script)



รูปที่ 2 แสดงถึงหน้าที่ของเชลล์

วิวัฒนาการของเชลล์

การพัฒนาของเชลล์ในระบบยูนิกซ์นั้นได้รับการพัฒนาปรับปรุงควบคู่กับระบบยูนิกซ์เองมาตลอด โดยมีลำดับขั้นดังนี้

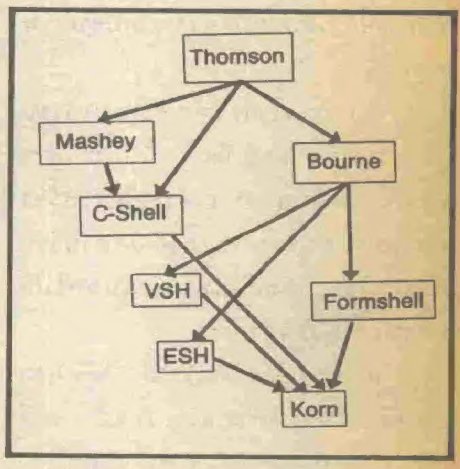
ทอมสัน เชลล์ (Thompson Shell) เป็นเชลล์ตัวแรกที่ใช้ในระบบยูนิกซ์ แต่ช่วงระยะเวลาในการใช้นั้นไม่มากนัก

มาร์เชย์ เชลล์ (Mashey Shell) เป็นเชลล์ที่ จอห์น มาร์เชย์ (John Mashey) และทีมงานได้ปรับปรุงเชลล์เดิม โดยเพิ่มขีดความสามารถในการทำงานเข้าไป อีกทั้งยังกำหนดให้มีการควบคุมทิศทางการไหลของโปรแกรม โดยมีความตั้งใจให้เชลล์นี้สามารถประพฤติตัวเสมือนภาษาโปรแกรมให้ได้

บอร์น เชลล์ (Bourne Shell) เป็นเชลล์ที่ สตีฟ บอร์น (Steve Bourne) ได้กำหนดโครงสร้างภาษาโปรแกรมขึ้นโดยมีลักษณะคล้าย ๆ กับภาษาอัลกอล (Algol) ไว้ในเชลล์ชนิดนี้ พร้อมทั้งยังให้มีการใช้ในลักษณะของโพรซีเดอร์ (Procedure) ด้วย

ซี เชลล์ (C Shell) เป็นเชลล์ที่ บิลล์ จอย (Bill Joy) แห่งเบอร์กเลย์ได้สร้างเชลล์ใหม่ขึ้นมาโดยมีลักษณะของไวยากรณ์คล้ายกับภาษาซี โดยสามารถควบคุมการไหลของข้อมูลและมีแฟ้มเก็บประวัติการใช้งานคำสั่งอีกด้วย

ซิสเต็มส์ไฟว์ เชลล์ (System V Shell) เป็นบอร์น เชลล์ (Bourne Shell) อีกเวอร์ชันหนึ่งที่เพิ่มฟังก์ชันการใช้งานต่าง ๆ เข้าไป



รูปที่ 3 วิวัฒนาการของเชลล์

V sh และ Esh เป็นบอร์น เชลล์ (Bourne Shell) เช่นเดียวกันแต่มีการเพิ่มเอดิเตอร์ vi กับ Emacs เข้าไป

คอร์น เชลล์ (Korn Shell) เป็นเชลล์ที่ออกแบบและพัฒนาโดยเดวิด จี

คอร์น (David G. Korn) แห่งศูนย์วิจัยเบลล์ โดยเชลล์นี้เป็นลักษณะของคำสั่งโต้ตอบ และคอร์น เชลล์มีขนาดใหญ่กว่าเชลล์อื่น ทำให้สามารถขยายขอบเขตและเพิ่มประสิทธิภาพการใช้งานได้มากยิ่งขึ้น การเขียนเชลล์สคริปต์ในคอร์นเชลล์นี้เขียนได้ง่ายและรัดกุม เป็นไปตามมาตรฐาน IEEE POSIX 1003.2 Shell and Utilities Language Committe

เลือกใช้เชลล์แบบไหนดี

เชลล์ที่ใช้กันอยู่ในขณะนี้มีอยู่ด้วยกัน 3 ชนิดคือ บอร์น เชลล์, ซี เชลล์ และ คอร์น เชลล์ ซึ่งแต่ละเชลล์ก็มีข้อดีข้อได้เปรียบแตกต่างกันไป ดังรูปที่ 4

บอร์น เชลล์ (sh) ถือว่าเป็นเชลล์มาตรฐานของระบบยูนิกซ์มีอยู่ในยูนิกซ์ทุกรุ่น สามารถย้ายเชลล์สคริปต์ข้ามเครื่องไปยังระบบยูนิกซ์ทุกรุ่นที่ใช้บอร์นเชลล์ได้

ซี เชลล์ (csh) มีขีดความสามารถสูงกว่าบอร์น เชลล์ มีแฟ้มเก็บคำสั่งการทำงาน แต่ไม่สามารถเข้าใจโครงสร้างของสคริปต์บอร์น เชลล์ได้ แต่สามารถเรียกบอร์น เชลล์ขึ้นมาเพื่อรันสคริปต์ของบอร์น เชลล์ได้

คอร์น เชลล์ (ksh) ได้ดึงเอาลักษณะเด่นของทั้งบอร์น เชลล์ และซี เชลล์เข้าด้วยกัน แต่คอร์น เชลล์นี้จะไม่มีในยูนิกซ์ทุกระบบ

การเลือกใช้เชลล์ว่าจะใช้ชนิดใดนั้นก็ต้องขึ้นอยู่กับระบบยูนิกซ์ที่ใช้งานอยู่ หากเป็นระบบรุ่นหลัง ๆ จะมีเชลล์ทั้ง 3 ชนิดให้เลือก ก็ควรเลือกใช้คอร์นเชลล์ เพราะเป็นเชลล์ที่ง่ายในการใช้งานและมีประสิทธิภาพสูงสุด แต่สำหรับระบบยูนิกซ์ที่ไม่มีคอร์น เชลล์ให้เลือก

Feature	sh	csh	ksh
Aliases		XX	XX
Bitwise operators		XX	
Built-in integer math		XX	XX
Built-in random numbers			XX
Built-in string operators			XX
Check for new mail	XX	XX	XX
Command directory hashing	XX	XX	
Command re-execution		XX	XX
Directory stacks		XX	
Exportable aliases			XX
File name completion		# #	
Formatted variables			XX
Functions	XX		XX
Interactive command-line editing			XX
Job control		# #	# #
Logout command file		XX	
Noclobber, Ignoreeof		XX	XX
Path for cd command	XX	XX	XX
Pathname modifiers		XX	
Select command			XX
Special prompt modifiers		XX	XX
Word lists/arrays		XX	XX
Note: # # means only available on some versions.			

รูปที่ 4 แสดงการเปรียบเทียบของเชลล์ทั้ง 3

นั้นก็ต้องแล้วแต่ความถนัดของผู้ใช้ว่าจะเลือกซี เชลล์ หรือบอร์น เชลล์ แต่ส่วนหลังในบทความนี้ที่จะกล่าวต่อไปจะอ้างถึงคอร์น เชลล์ซึ่งเป็นเชลล์ที่ดีที่สุดขณะนี้

การทำงานของเชลล์

ยูนิกซ์เป็นโอเอสแบบมัลติทาสกิง หรือบางครั้งก็เรียกว่ามัลติโปรเซสซิง ซึ่งหมายถึงการทำงานได้หลาย ๆ งานในเวลาเดียวกัน แต่เชลล์ 1 เชลล์นั้นจะรองรับการทำงานได้เพียงงานเดียวเท่านั้น ดังนั้นในยูนิกซ์ต้องมีกลวิธีที่จะทำให้ทำงานพร้อม ๆ กันหลาย ๆ งานได้โดยไม่ต้องรอให้งานอื่นเสร็จก่อน เพราะจะทำให้สิ้นเปลืองเวลา จึงจะเป็นข้อได้เปรียบของโอเอสประเภทนี้

การจัดการในระบบยูนิกซ์ในขุดต้น ๆ นั้นการทำงานของคำสั่งจะมี 2 ลักษณะคือ

1. synchronous execution เป็นการทำงานตามคำสั่งจนจบสิ้นก่อนจึงจะพร้อมรับคำสั่งต่อไป เรียกว่า การทำงานเบื้องหน้า (foreground mode)

2. asychronous execution เป็นการทำงานตามคำสั่งที่กำหนดพร้อมกับสร้างเชลล์ใหม่ขึ้นมารับคำสั่งต่อไป โดย

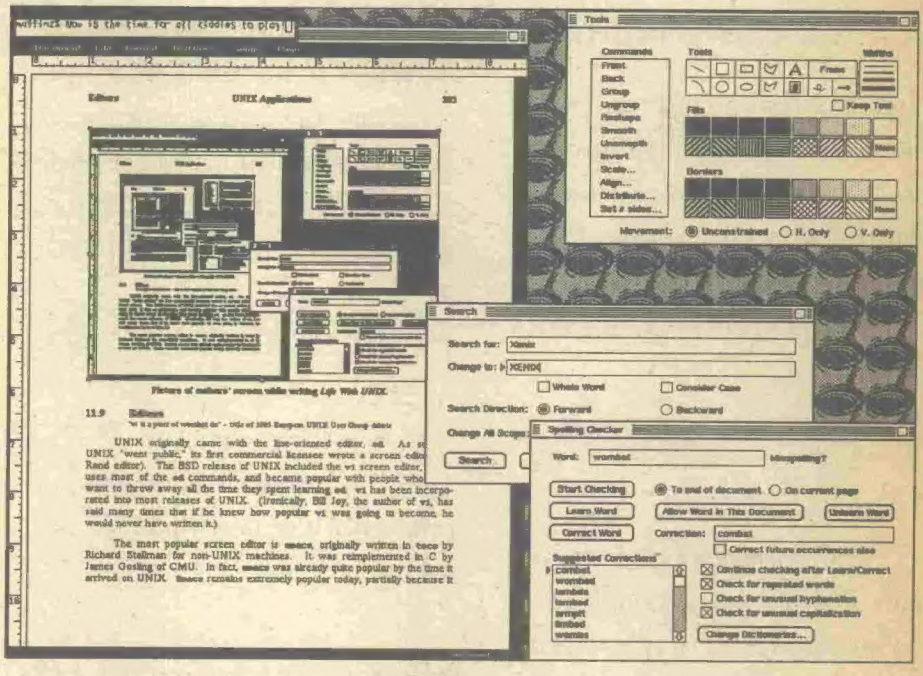
ไม่ต้องรอให้การทำงานในคำสั่งนั้นเสร็จสิ้น การทำงานในลักษณะนี้เรียกว่า การทำงานเบื้องหลัง (background mode) เพียงแต่เติมเครื่องหมาย ampersand (&) ท้ายคำสั่ง

การทำงานเบื้องหลังสามารถทำได้หลาย ๆ งานส่วนการทำงานเบื้องหน้านั้นจะมีได้เพียงหนึ่งงานเท่านั้น เพราะหากมีงานเบื้องหน้าหลายงานแล้ว จะต้องเกิดปัญหาความสับสนว่าผู้ใช้ต้องการติดต่อกับงานใดและไม่สามารถสลับเปลี่ยนระหว่างงานเบื้องหน้ากับงานเบื้องหลังได้ ต่อมาในระบบยูนิกซ์ได้พัฒนาวิธีการต่าง ๆ ที่ใช้ในการควบคุมการทำงานให้มีความอ่อนไหวมากขึ้น โดยวิธีที่นิยมใช้กันอย่างมาก 2 วิธีคือ วิธีควบคุมงาน (job control) และวินโดว์ (windows)

วิธีควบคุมงานเป็นวิธีที่ให้ระบบยูนิกซ์สามารถกระทำหรือพักการทำงานของโปรเซส รวมถึงการติดต่อกับอินพุตและเอาต์พุตด้วย โดยการควบคุมงานทำให้มีการสแวป (swap) ระหว่างงานเบื้องหน้ากับงานเบื้องหลังได้ โดยการสแวปโปรเซสนี้อาจเป็นการใช้ปุ่มควบคุมเป็นคำสั่งในการจัดการ แต่ลักษณะของวิธีควบคุมงานนั้นจะไม่สามารถเก็บสภาวะต่าง ๆ ได้หมดในสภาวะพักงาน (suspend)

วินโดว์เป็นลักษณะของการแบ่งสกรีนของจอภาพออกเป็นส่วน ๆ โดยเรียกว่าวินโดว์ (windows) โดยแต่ละวินโดว์สามารถรันโปรเซสได้เป็นอิสระต่อกัน ถือได้ว่าทุกโปรเซสในแต่ละวินโดว์เป็นโปรเซสเบื้องหน้า (ดูรูปที่ 5)

สำหรับวินโดว์ก็ยังประสบปัญหากับว่าโปรเซสไหนผู้ใช้ต้องการติดต่อด้วย จึงต้องทำการตรวจเช็คสัญญาณ



รูปที่ 5 แสดงการเปิดหลาย ๆ วินโดว์บนสกรีน

mouse stoke วินโดว์เป็นระบบที่นิยมอย่างมากในปัจจุบัน เพราะสามารถเปิดหลายวินโดว์เพื่อใช้ในการคีบัก, เอดิเตอร์ หรือใช้คอมพายล์โปรแกรม แต่สิ่งที่ต้องสูญเสียก็คือ ค่าใช้จ่ายสูงสำหรับอุปกรณ์ต่าง ๆ และที่สำคัญจะต้องมีการแบ่งเวลาการทำงานของซีพียูออกไปอีก

รูปแบบคำสั่ง

เชลล์เมื่อเทียบกับดอสแล้วได้เท่ากับ command.com แต่เชลล์มีขีดความสามารถสูงกว่ามาก โดยตัวเชลล์สามารถเป็นได้ทั้งตัวแปรคำสั่งและตัวกำหนดการเปลี่ยนทิศทางผลลัพธ์ (Redirection)

คำสั่งในระบบยูนิกซ์มีลักษณะเหมือนกับคำสั่งทั่ว ๆ ไปกับระบบอื่น คือ จะมีการรับข้อมูลจากอินพุตมาตรฐานคือคีย์บอร์ด และแสดงผลลัพธ์ออกทางเอาต์พุตมาตรฐานคือจอเทอร์มินัล และรายงานผลของความผิดพลาดทางจอเทอร์มินัล เช่น คำสั่ง who เป็นคำสั่งให้แสดงรายชื่อของผู้ใช้งานในระบบรวมทั้งหมายเลขเทอร์มินัลและเวลาเริ่มต้นเข้าสู่ระบบ ดังรูปที่ 6

```
$ who
    aew         tty01 APR11 11:30
    pre         tty04 APR11 12:14
    tam         tty05 APR11 12:34
    mam         tty06 APR11 10:10
    moobin      tty08 APR11 12:17
```

รูปที่ 6

การเปลี่ยนทิศทางผลลัพธ์

ในระบบยูนิกซ์ได้มีการพัฒนาให้มีขีดความสามารถที่จะเปลี่ยนทิศทางการแสดงผลลัพธ์ได้โดยอาจจะนำผลลัพธ์ไปเก็บในไฟล์ ๆ หนึ่ง แทนที่จะออกทางจอภาพเลย โดยมีเครื่องหมาย > เป็นตัวกำหนดทิศทางแล้วตามด้วยชื่อไฟล์ที่ใช้เก็บ

```
$ who > users
```

เป็นการนำผลลัพธ์จากคำสั่ง who ไปเก็บที่ไฟล์ชื่อ users โดยอยู่ในส่วนของไดเรคทอรีที่ใช้งานอยู่ในขณะนั้น ลองแสดงผลของไฟล์ users ดูเพื่อตรวจสอบว่าเป็นไปตามที่คิดหรือไม่ โดยใช้คำสั่ง cat ตามด้วยชื่อไฟล์ และต้องอยู่ในไดเรคทอรีเดิมด้วย ดังรูปที่ 7

```
$ cat users
  aew      tty01 APR11 11:30
  pre      tty04 APR11 12:14
  tam      tty05 APR11 12:34
  mam      tty06 APR11 10:10
  moobin   tty08 APR11 12:17
```

รูปที่7

การเปลี่ยนทิศทางผลลัพธ์โดยใช้เครื่องหมาย > นั้นหากมีชื่อไฟล์นั้นอยู่ก่อนหน้านี้แล้ว ผลลัพธ์ของคำสั่งใหม่จะมาแทนที่ข้อมูลในไฟล์เดิมหมด หากต้องการเพียงแต่ให้ผลลัพธ์ของคำสั่งใหม่มาต่อท้ายไฟล์เดิม ก็เพียงแต่กำหนดเครื่องหมายเปลี่ยนทิศทางจาก > เป็น >> นอกจากนี้เรายังสามารถนำลักษณะของการเปลี่ยนทิศทางมาใช้ในการส่งค่าอินพุตแล้วให้ผลลัพธ์เป็นไปตามคำสั่งพร้อมทั้งกำหนดทิศทางได้ด้วย เช่น

```
$ sort < users > sorted__users
```

จากตัวอย่างจะเป็นการใช้ไฟล์ users เป็นอินพุตของคำสั่ง sort แล้วนำผลลัพธ์ที่ได้ไปเก็บไว้ที่ไฟล์ชื่อ sorted__users

```
$ cat sorted__users.
  aew      tty01 APR11 11:30
  mam      tty08 APR11 12:17
  moobin   tty06 APR11 10:10
  pre      tty04 APR11 12:14
  tam      tty05 APR11 12:34
```

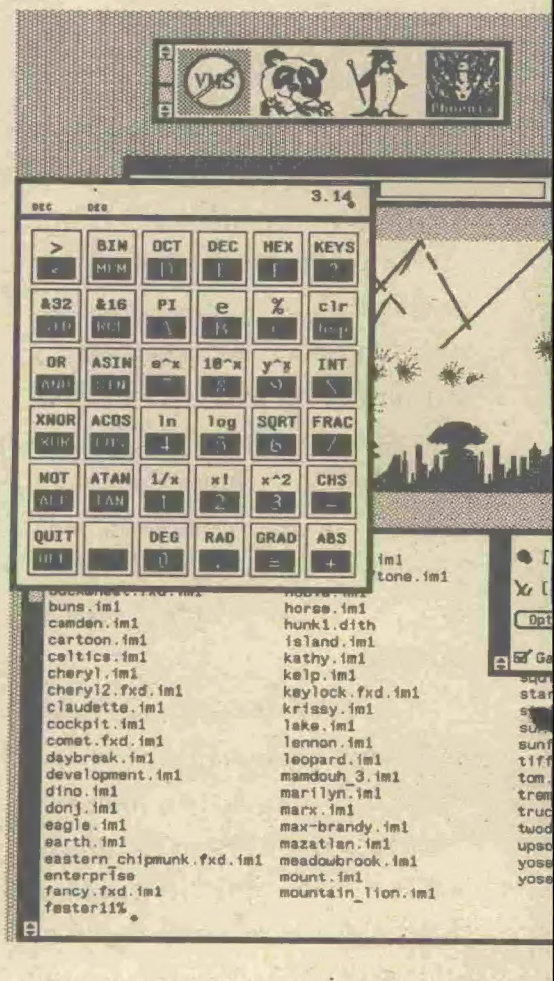
รูปที่ 8

การเชื่อมต่อท่อ (Piping)

เมื่อสังเกตจะเห็นได้ว่า รูปแบบของการเปลี่ยนทิศทางจะต้องนำผลลัพธ์ไปเก็บไว้ในไฟล์ ๆ หนึ่งก่อน ซึ่งทำให้สิ้นเปลืองเนื้อที่ของหน่วยความจำไปเท่ากับขนาดไฟล์ที่เก็บ ดังนั้นในระบบยูนิกซ์จึงสร้างลักษณะพิเศษขึ้นมาเป็นการเชื่อมต่อคำสั่งเข้าด้วยกัน คือ ไปป์ (pipe) โดยมีลักษณะการทำงานคล้าย ๆ กับการนำท่อน้ำมาต่อเข้าด้วยกัน ดังรูปที่ 9

```
$ who >  users
$ wc-l >  users การใช้ในรูป
          แบบเปลี่ยนทิศทาง
      5
$ who :  wc-l การใช้ไปป์
      5
```

รูปที่ 9



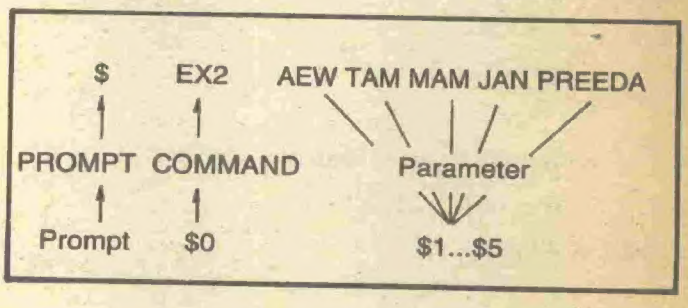
เชลล์โปรแกรมมิง

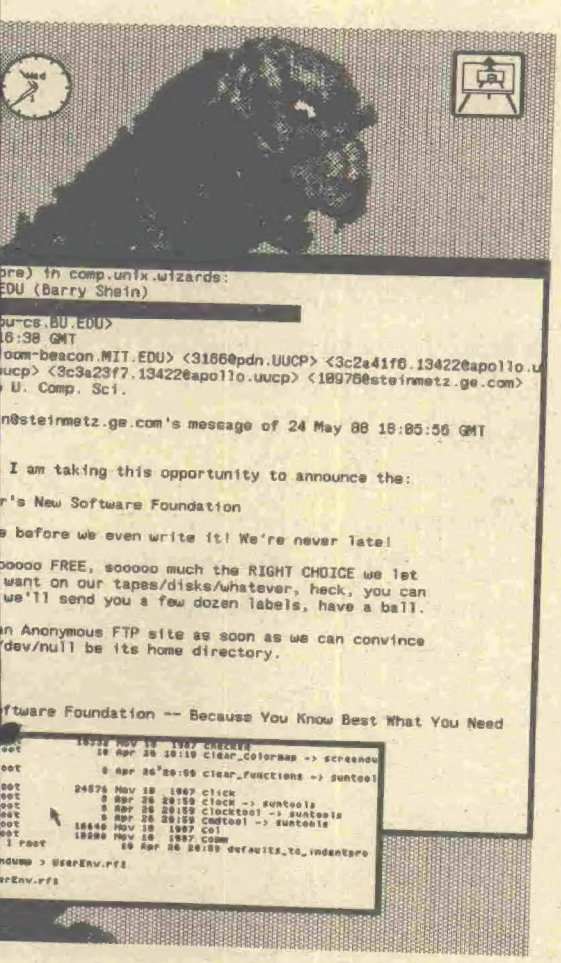
เชลล์โปรแกรมมิงหรือเชลล์สคริปต์เป็นขีดความสามารถอันสำคัญที่เพิ่มเข้ามาในตัวเชลล์ทำให้เชลล์ทำงานที่ซับซ้อนได้ยิ่งขึ้น โดยเฉพาะอย่างยิ่งเชลล์สคริปต์ของคอร์น เชลล์แล้ว มีทั้งการกำหนดตัวแปร การควบคุมทิศทางการไหล และมีรูปแบบการสร้างโปรแกรมได้ง่าย

รูปที่ 10

$0	เป็นส่วนของตัวคำสั่งเอง
$1...$9	เป็นส่วนของตัวแปรตำแหน่ง
$#	จำนวนพารามิเตอร์ในบรรทัดคำสั่งนั้น
$*	รวมตัวแปรทุกตำแหน่ง

รูปที่ 11





ทำไมต้องเขียนเชลล์สคริปต์

1) การเขียนเชลล์สคริปต์ ทำให้ไม่ต้องพิมพ์คำสั่งหลาย ๆ คำสั่ง ซ้ำ ๆ ซาก ๆ และยังป้องกันความผิดพลาดในการพิมพ์ได้อีกด้วย

2) รูปแบบของเชลล์สคริปต์นั้นมีส่วนควบคุมการทำงานเหมือนกับภาษาโปรแกรมภาษาหนึ่ง ดังนั้นการใช้เชลล์สคริปต์สามารถทำงานต่างกันได้ขึ้นอยู่กับสภาพข้อมูลหรือคำสั่งเพิ่มเติม

เขียนเชลล์สคริปต์ด้วยวิธีไหน

เชลล์สคริปต์เป็นโครงสร้างแบบเท็กซ์ทั่วไป เพียงแต่มีการกำหนดตัวแปรและนำโครงสร้างการควบคุมการไหลมาประกอบเข้าด้วยกัน ดังนั้นวิธีที่ง่ายที่สุดคือการเขียนเชลล์สคริปต์ผ่านทางเอดิเตอร์ ซึ่งในระบบยูนิกซ์ทั่วไปจะมี vi editor ในระบบอยู่แล้ว จึงเป็นการสะดวกในการเขียนเชลล์สคริปต์

การเรียกใช้เชลล์สคริปต์

การใช้งานเชลล์สคริปต์นั้นสามารถทำได้ 2 วิธีด้วยกันคือ

1) ใช้เชลล์เป็นตัวดำเนินตามคำสั่ง โดยมีรูปแบบเป็น

```
$ ksh <ชื่อเชลล์สคริปต์>
```

2) ทำเชลล์ให้ใช้งานได้โดยอัตโนมัติ โดยใช้คำสั่ง chmod

```
$ chmod + x <ชื่อเชลล์สคริปต์>
```

เมื่อถึงขั้นตอนนี้แล้ว เชลล์สคริปต์สามารถใช้งานได้โดยอัตโนมัติคือเรียกชื่อเชลล์สคริปต์นั้นได้เลย

```
$ <ชื่อเชลล์สคริปต์>
```

ตัวแปรพิเศษและตัวแปรชั่วคราว

ตัวแปรในเชลล์สคริปต์ทุกตัวจะเริ่มต้นด้วย $ แล้วตามด้วยอักขระตัวอื่น ตัวแปรพิเศษในเชลล์จะเป็นดังรูปที่ 10 และตัวอย่างดังรูปที่ 11

ลองเขียนเชลล์สคริปต์ง่าย ๆ ดู โดยตั้งชื่อว่า EX2 ดังรูปที่ 12

```
echo "The Script name is $0"
echo "$1 and $2"
echo "$4 and $5"
```

รูปที่ 12

เราจะได้ผลลัพธ์เมื่อเรียกใช้สคริปต์ EX2 AEW TAM MAM JAN PREEDA ดังรูปที่ 13

```
The Script name is EX2
AEW and TAM
JAN and PREEDA
```

รูปที่ 13

จะเห็นว่า MAM ไม่ได้เข้ามาเกี่ยวข้องเนื่องจาก MAM คือ $3 ซึ่งไม่ได้ถูกเรียกใช้และถึงแม้ตัวแปรตำแหน่งที่เห็นมีเพียง $1...$9 นั้น เราก็ยังสามารถกำหนดตัวแปรได้เกิน 9 ตัวโดยใช้คำสั่ง shift เข้าช่วย โดยจะมีการเลื่อนตัวแปรไปทางขวา 1 ตำแหน่ง นอกจากนี้ยังสามารถเก็บค่าตัวแปรชั่วคราวได้ โดยการกำหนดชื่อตัวแปรและค่าที่ต้องการให้เป็นได้โดยตรง

```
<ชื่อตัวแปรชั่วคราว>
= <ข้อความที่เก็บ>
```

ส่วนการเรียกข้อความกลับมาทำได้โดย

```
$ <ชื่อตัวแปรชั่วคราว>
```

รูปที่ 14

```
one = "AEW"
two = "TAM"
echo"$one asked $two : What's happening?"
echo"$two replied : I think we are being called."
echo"$one I think too he said $# words__$*"
```

รูปที่ 15

```
AEW asked TAM : What's happening"
TAM replied : I think we are being called.
AEW I think too he said 4 words Close book
to sleep
```

สร้างเชลล์สคริปต์ชื่อ All ดังรูปที่ 14

เมื่อเรียกเชลล์สคริปต์นี้โดยใช้คำสั่ง

```
$ All Close book to sleep
```

จะได้ผลลัพธ์ดังรูปที่ 15

สรุป

เชลล์เป็นส่วนหนึ่งที่ทำให้ระบบยูนิกซ์มีประสิทธิภาพสูงขึ้นเป็นอย่างมาก แต่เชลล์ที่กล่าวนี้เป็นเพียงส่วนย่อยเท่านั้น เพื่อให้มองเห็นหน้าที่และการทำงานของเชลล์อย่างคร่าว ๆ ส่วนลักษณะของสัญลักษณ์พิเศษต่าง ๆ และโครงสร้างทางโปรแกรมมิงของเชลล์จะขอกล่าวในฉบับหน้า สวัสดีครับ